18 | 实战痛点4：Vue 3 项目中的性能优化

2021-11-26 大圣

《玩转Vue 3全家桶》 课程介绍 >

讲述：大圣

时长 13:51 大小 12.69M

你好，我是大圣，欢迎进入课程的第 18 讲。

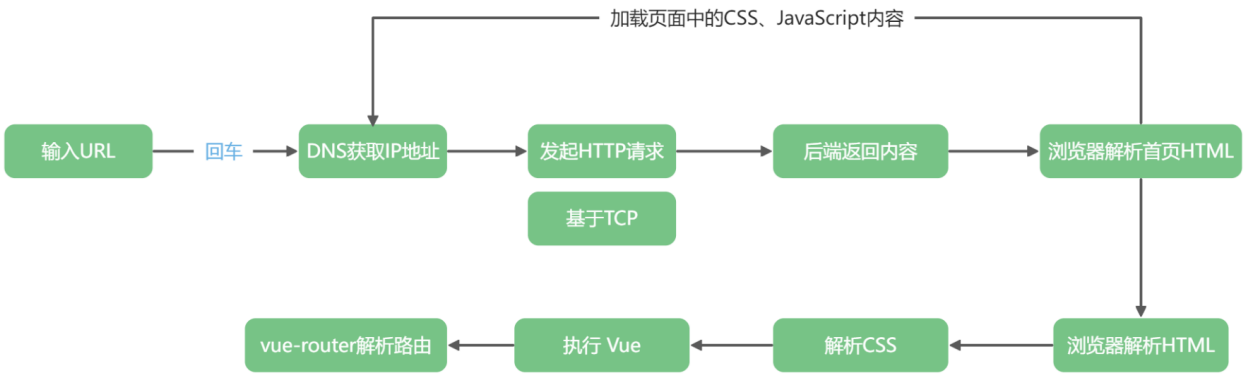
在上一讲中，我们聊了项目中的权限设计，相信你对 Vue 项目中如何设计，以及如何实现权限都有了自己的心得。今天，我们来聊一下 Vue 项目中的另外一个难点：性能优化。性能优化是一个老生常谈的话题，如果你是前端从业者，那么无论是求职的简历，还是晋升的 PPT，性能优化相关的内容都是不可或缺的。

那么在 Vue 项目中，我们应该如何做性能优化呢？下面，我们会先从 Vue 项目在整体上的执行流程谈起，然后详细介绍性能优化的两个重要方面：**网络请求优化和代码效率优化**。不过，在性能优化之外，用户体验才是性能优化的目的，所以我也会简单谈一下用户体验方面的优化项。最后，我还会通过性能监测报告，为你指引出性能优化的方向。

用户输入 URL 到页面显示的过程

我们先来聊一个常见的面试题，那就是用户从输入 URL，然后点击回车，到页面完全显示出来，这一过程中到底发生了什么？

通过下图，我们可以从前端的视角看到从输入 URL 到页面显示的大致过程：



简单来说，就是用户在输入 URL 并且敲击回车之后，浏览器会去查询当前域名对应的 IP 地址。对于 IP 地址来说，它就相当于域名后面的服务器在互联网世界的门牌号。然后，浏览器会向服务器发起一个网络请求，服务器会把浏览器请求的 HTML 代码返回给浏览器。

之后，浏览器会解析这段 HTML 代码，并且加载 HTML 代码中需要加载的 CSS 和 JavaScript，然后开始执行 JavaScript 代码。进入到项目的代码逻辑中，可以看到 Vue 中通过 vue-router 计算出当前路由匹配的组件，并且把这些组件显示到页面中，这样我们的页面就完全显示出来了。而我们性能优化的主要目的，就是让页面显示过程的时间再缩短一些。

性能优化

从用户输入 URL 到页面显示的过程这个问题，包含着项目页面的执行流程。这个问题之所以重要，是因为我们只有知道了在这个过程中，每一步都发生了什么，之后才能针对每一步去做网络请求的优化，这也是性能优化必备的基础知识。

网络请求优化

对于前端来说，可以优化的点，首先就是在首页的标签中，使用标签去通知浏览器对页面中出现的其他域名去做 DNS 的预解析，比如页面中的图片通常都是放置在独立的 CDN 域名下，这样页面加载首页的时候就能预先解析域名并把结果缓存起来 。

因为极客时间首页没做这个优化，所以我们以淘宝网的首页为例进行分析。你可以在淘宝的首页源码中看到下图所示的一列 dns-prefetch 标签，这样首页再出现 img.alicdn.com 这个域名请求的时候，浏览器就可以从缓存中直接获取对应的 IP 地址。

```
<meta name="aplus-xplug" content="NONE">
<meta name="keyword" content="淘宝,掏宝,网上购物,C2C,在线交易,交易市场,网上交易,交易市场,网
<link rel="dns-prefetch" href="//g.alicdn.com" />
<link rel="dns-prefetch" href="//img.alicdn.com" />
<link rel="dns-prefetch" href="//tce.alicdn.com" />
<link rel="dns-prefetch" href="//gm.mmstat.com" />
<link ref="dns-prefetch" href="//tce.taobao.com" />
<link rel="dns-prefetch" href="//log.mmstat.com" />
<link rel="dns-prefetch" href="//tui.taobao.com" />
<link rel="dns-prefetch" href="//ald.taobao.com" />
<link rel="dns-prefetch" href="//gw.alicdn.com" />
<link rel="dns-prefetch" href="//atanx.alicdn.com"/>
<link rel="dns-prefetch" href="//dfhs.tanx.com"/>
<link rel="dns-prefetch" href="//ecpm.tanx.com" />
<link rel="dns-prefetch" href="//res.mmstat.com" />
<link href="//img.alicdn.com/tps/i3/T1OjaVFl4dXXa.JOZB-114-114.png" rel="apple-touc
```

项目在整体流程中，会通过 HTTP 请求加载很多的 CSS、JavaScript，以及图片等静态资源。为了让这些文件在网络加载中更快，我们可以从后面这几方面入手进行优化。

首先，浏览器在获取网络文件时，需要通过 HTTP 请求，HTTP 协议底层的 TCP 协议每次创建链接的时候，都需要三次握手，而三次握手会造成额外的网络损耗。如果浏览器需要获取的文件较多，那就会因为三次握手次数过多，而带来过多网络损耗的问题。

所以，首先我们需要的是让文件尽可能地少，这就诞生出一些常见的优化策略，比如先给文件打包，之后再上线；使用 CSS 雪碧图来进行图片打包等等。文件打包这条策略在 HTTP2 全面普及之前还是有效的，但是在 HTTP2 普及之后，多路复用可以优化三次握手带来的网络损耗。关于 HTTP2 的更多内容，你可以去搜索相关文章自行学习。

其次，**除了让文件尽可能少，我们还可以想办法让这些文件尽可能地小一些**，因为如果能减少文件的体积，那文件的加载速度自然也就会变快。这一环节也诞生出一些性能优化策略，比如 CSS 和 JavaScript 代码会在上线之前进行压缩；在图片格式的选择上，对于大部

分图片来说，需要使用 JPG 格式，精细度要求高的图片才使用 PNG 格式；优先使用 WebP 等等。也就是说，尽可能在同等像素下，选择体积更小的图片格式。

在性能优化中，懒加载的方式也被广泛使用。图片懒加载的意思是，我们可以动态计算图片的位置，只需要正常加载首屏出现的图片，其他暂时没出现的图片只显示一个占位符，等到页面滚动到对应图片位置的时候，再去加载完整图片。

除了图片，项目中也会做路由懒加载，现在项目打包后，所有路由的代码都在首页一起加载。但是，我们也可以把不常用的路由单独打包，在用户访问到这个路由的时候再去加载代码。下面的代码中，vue-router 也提供了懒加载的使用方式，只有用户访问了 /course/:id 这个页面后，对应页面的代码才会加载执行。

复制代码

```
{
  path: '/course/:id',
  component: () => import('../pages/courseInfo'),
},
```

在文件大小的问题上，Lighthouse 已经给了我们比较详细的优化方法，比如控制图片大小、减少冗余代码等等，我们可以在项目打包的时候，使用可视化的插件来查看包大小的分布。

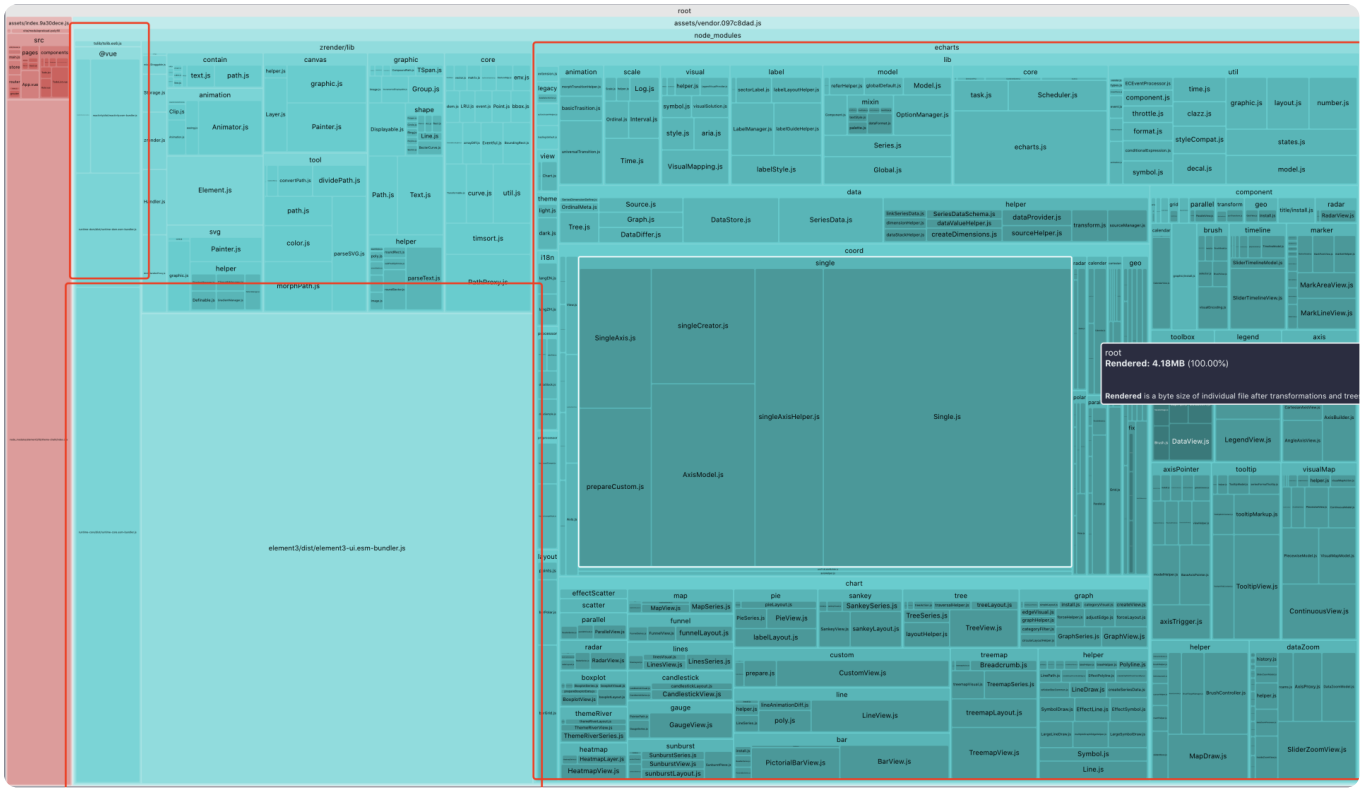
我们来到项目根目录下，通过执行 npm install 操作来安装插件 rollup-plugin-visualizer。使用这个插件后，我们就可以获取到代码文件大小的报告了。之后，进入到 vite.config.js 这个文件中，新增下列代码，就可以在 Vite 中加载可视化分析插件。

复制代码

```
import { visualizer } from 'rollup-plugin-visualizer'
export default defineConfig({
  plugins: [vue(),vueJsx(), visualizer()],
})
```

然后，我们在项目的根目录下执行 npm run build 命令后，项目就把项目代码打包在根目录的 dist 目录下，并且根目录下多了一个文件 stat.html。

我们用浏览器打开这个 stat 文件，就能看到下面的示意图。项目中的 ECharts 和 Element3 的体积远远大于项目代码的体积，这时候我们就需要用懒加载和按需加载的方式，去优化项目整体的体积。



那么这些文件如何才能高效复用呢？**我们需要做的，就是尽可能高效地利用浏览器的缓存机制，在文件内容没有发生变化的时候，做到一次加载多次使用**，项目中如果成功复用一个几百 KB 的文件，对于性能优化来说是一个巨大的提升。

浏览器的缓存机制有好几个 Headers 可以实现，Expires、Cache-control，last-modify、etag 这些缓存相关的 Header 可以让浏览器高效地利用文件缓存。我们需要做的是，只有当文件的内容修改了，我们才会重新加载文件。这也是为什么我们的项目执行 npm run build 命令之后，静态资源都会带上一串 Hash 值，因为这样确保了只有文件内容发生变化的时候，文件名才会发生变化，其他情况都会复用缓存。

代码效率优化

在浏览器加载网络请求结束后，页面开始执行 JavaScript，因为 Vue 已经对项目做了很多内部的优化，所以在代码层面，我们需要做的优化并不多。很多 Vue 2 中的性能优化策略，在 Vue 3 时代已经不需要了，我们需要做的就是**遵循 Vue 官方的最佳实践**，其余的交给 Vue 自身来优化就可以了。

比如 computed 内置有缓存机制，比使用 watch 函数好一些；组件里也优先使用 template 去激活 Vue 内置的静态标记，也就是能够对代码执行效率进行优化；v-for 循环渲染一定要有 key，从而能够在虚拟 DOM 计算 Diff 的时候更高效复用标签等等。然后就是 JavaScript 本身的性能优化，或者说某些实现场景算法的选择了，这里需要具体问题具体分析，在通过性能监测工具发现代码运行的瓶颈后，我们依次对耗时过长的函数进行优化即可。

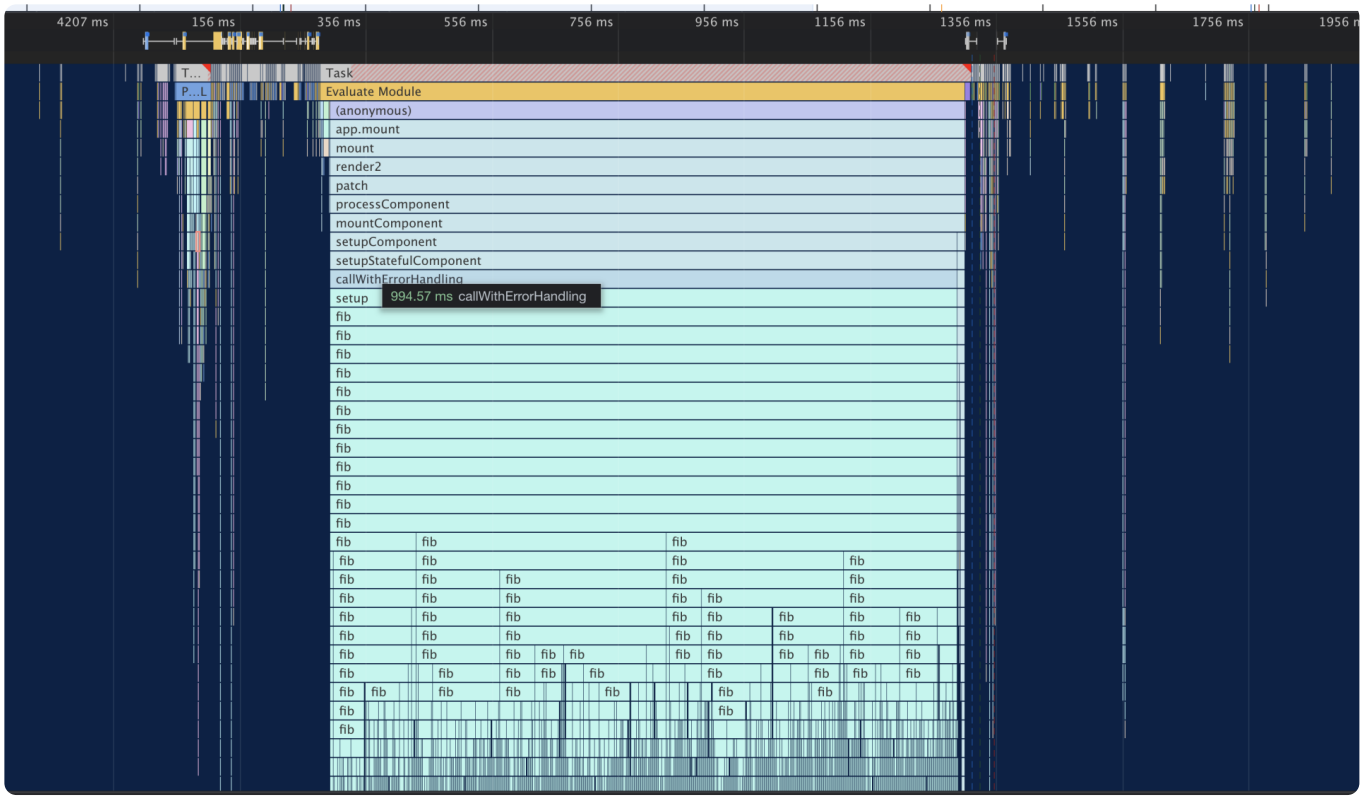
我们来到 src/App.vue 文件中，看下面的代码，我们实现了一个斐波那契数列，也就是说，在我们实现的这个数列中，每一个数的值是前面两个数的值之和。我们使用简单的递归算法实现斐波那契数列后，在页面显示计算结果。

复制代码

```
function fib(n){
  if(n<=1) return 1
  return fib(n-1)+fib(n-2)
}
let count = ref(fib(38))
```

上面的代码在功能上，虽然实现了斐波那契数列的要求，但是我们能够感觉到页面有些卡顿，所以我们来对页面的性能做一下检测。

我们打开调试窗口中的 Performance 面板，使用录制功能后，便可得到下面的火焰图。通过这个火焰图，我们可以清晰地定位出这个项目中，整体而言耗时最长的 fib 函数，并且我们能看到这个函数被递归执行了无数次。到这里，我们不难意识到这段代码有性能问题。不过，定位到问题出现的地方之后，代码性能的优化就变得方向明确了。



下面的代码中，我们使用递推的方式优化了斐波那契数列的计算过程，页面也变得流畅起来，这样优化就算完成了。其实对于斐波那契数列的计算而言，得到最好性能的方式是使用数学公式 + 矩阵来计算。不过在项目瓶颈到来之前，我们采用下面的算法已经足够了，**这也是性能优化另外一个重要原则，那就是不要过度优化。**

复制代码

```
function fib(n){
  let arr = [1,1]
  let i = 2
  while(i<=n){
    arr[i] = arr[i-1]+arr[i-2]
    i++
  }
  return arr[n]
}
```

用户体验优化

性能优化的主要目的，还是为了能让用户在浏览网页的时候感觉更舒服，所有有些场景我们不能只考虑单纯的性能指标，还要结合用户的交互体验进行设计，**必要的时候，我们可以损失一些性能去换取交互体验的提升。**

比如用户加载大量图片的同时，如果本身图片清晰度较高，那直接加载的话，页面会有很多图一直是白框。所以我们也可以预先解析出图片的一个模糊版本，加载图片的时候，先加载这个模糊的图作为占位符，然后再去加载清晰的版本。虽然额外加载了图片文件，但是用户在体验上得到了提升。

类似的场景还有很多，比如用户上传文件的时候，如果文件过大，那么上传可能就会很耗时。而且一旦上传的过程中发生了网络中断，那上传就前功尽弃了。

为了提高用户的体验，我们可以选择断点续传，也就是把文件切分成小块后，挨个上传。这样即使中间上传中断，但下次再上传时，只上传缺失的那些部分就可以了。可以看到，断点上传虽然在性能上，会造成网络请求变多的问题，但也极大地提高了用户上传的体验。

还有很多组件库也会提供骨架图的组件，能够在页面还没有解析完成之前，先渲染一个页面的骨架和 loading 的状态，这样用户在页面加载的等待期就不至于一直白屏，下图所示就是 antd-vue 组件库骨架图渲染的结果。

性能监测报告

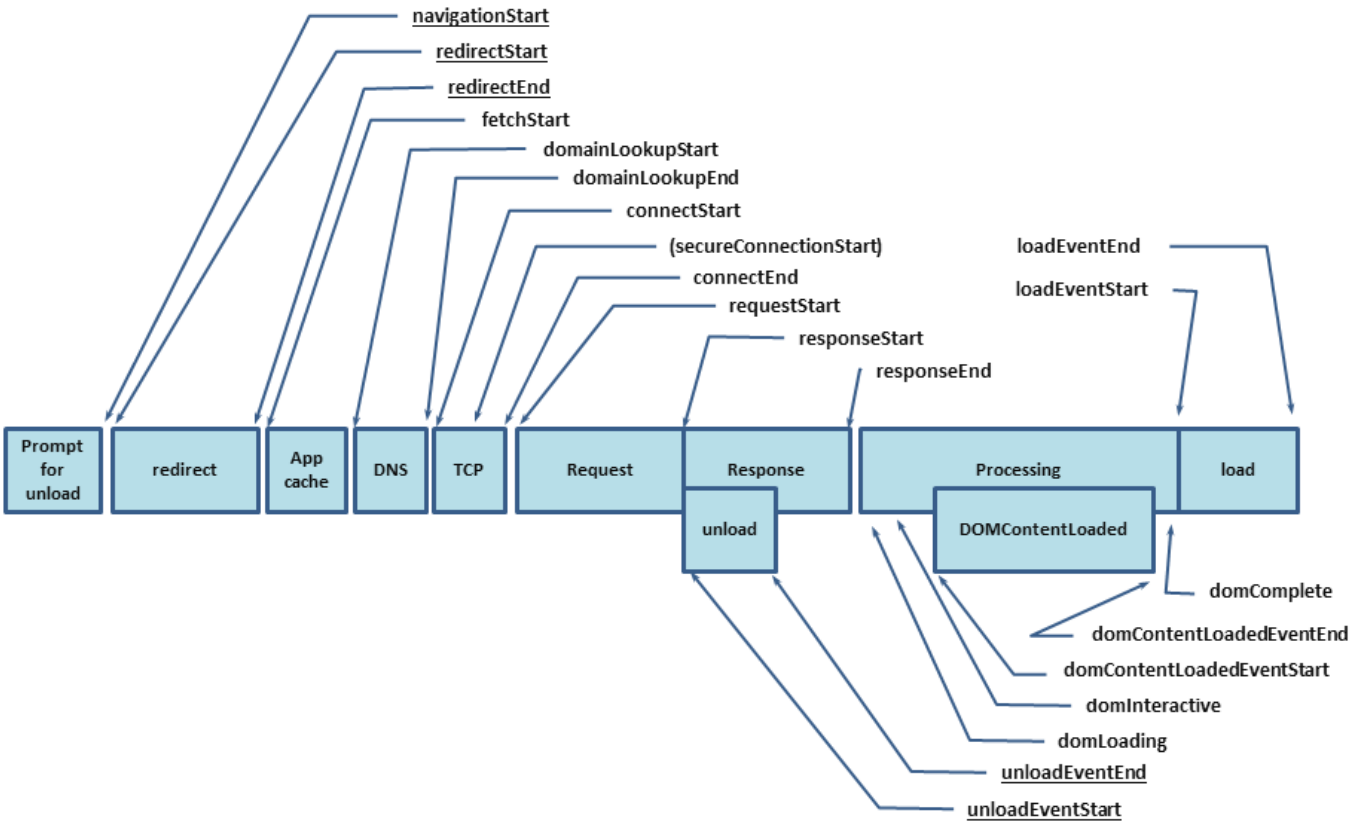
在第 12 讲学习 Vue Devtools 的时候，我们已经使用 Chrome 的性能监测工具 Lighthouse 对极客时间的官网做了一次性能的评估，我们可以在这里看到评测报告。并

且，我们也对如何在调试窗口的 Performance 页面中进行性能监控，给出了演示。为了方便你理解，我们在这里也解释一下 FCP、TTI 和 LCP 这几个关键指标的含义。

首先是 First Contentful Paint，通常简写为 FCP，它表示的是页面上呈现第一个 DOM 元素的时间。在此之前，页面都是白屏的状态；然后是 Time to interactive，通常简写为 TTI，也就是页面可以开始交互的时间；还有和用户体验相关的 Largest Contentful Paint，通常简写为 LCP，这是页面视口上最大的图片或者文本块渲染的时间，在这个时间，用户能看到渲染基本完成后的首页，这也是用户体验里非常重要的一个指标。

我们还可以通过代码中的 performance 对象去动态获取性能指标数据，并且统一发送给后端，实现网页性能的监控。性能监控也是大型项目必备的监控系统之一，可以获取到用户电脑上项目运行的状态。

下图展示了 performance 中所有的性能指标，我们可以通过这些指标计算出需要统计的性能结果。



复制代码

```
let timing = window.performance && window.performance.timing
let navigation = window.performance && window.performance.navigation

DNS 解析：
```

```
let dns = timing.domainLookupEnd - timing.domainLookupStart

总体网络交互耗时：
let network = timing.responseEnd - timing.navigationStart

渲染处理：
let processing = (timing.domComplete || timing.domLoading) - timing.domLoading

可交互：
let active = timing.domInteractive - timing.navigationStart
```

在上面的代码中，我们通过 Performance API 获取了 DNS 解析、网络、渲染和可交互的时间消耗。有了这些指标后，我们可以随时对用户端的性能进行检测，做到提前发现问题，提高项目的稳定性。

总结

今天的主要内容就聊完啦，我们来复习一下今天学到的内容吧。

首先我们了解了用户从输入 URL 到页面显示的这一过程发生了什么，这里面的每个流程都有值得优化的地方，比如网络请求、页面渲染等。在对这些流程优化后，网页运行时整体的性能都会得到提升。

之后，在网络请求优化这一部分，我们首先谈到，对于 DNS，我们可以通过 dns-prefetch 预先获取，这对性能优化来说，会减少页面中其他域名请求的 DNS 解析时间；因为 TCP 协议每次链接时，都需要三次握手，而这会带来额外的网络消耗的问题，为了解决这一问题，我们的优化策略是让文件尽可能少一些，并且也小一些。

比如，我们可以通过文件打包的形式减少 HTTP 请求数量，这样对于文件的大小来说，可以减小文件体积。我们也可以压缩代码，以及选择更合适的图片格式，这些都可以让我们加载更小的文件。图片的懒加载和路由的懒加载可以让首页加载更少的文件，从而实现页面整体性能的优化。

在讲完网络请求优化后，我们又研究了代码效率优化这个问题，其实代码层面要做的优化并不多，主要还是遵守 Vue 3 最佳实践。我们还以斐波那契数列的计算为例，通过在 Performance 面板中进行性能监控，明确了代码优化的方向。在通过递归的方式优化斐波那契数列之后，我们能明白这样一点：**性能优化的一个重要原则，是不要过度优化。**

之后，在用户体验优化这一部分，我们的关注点是在交互体验的优化上。有些场景我们可以损失部分性能去换取体验的提升，比如通过骨架图，我们可以在页面加载之前，通过对图片预先加载出模糊版本，可以让用户获得更好的体验。

最后，在性能监测报告这一部分，我讲到选择合适的工具，可以帮助我们实时地监测项目的性能。我们通过 Lighthouse 性能报告和 Performace 监测工具，可以精确地定位到项目瓶颈所在，有针对地去进行性能优化。

思考题

最后，给你留一个思考题：通过今天的学习，想一想你负责的项目都有哪些可以优化的地方呢？

欢迎在留言区留言分享你的想法，也欢迎你把这一讲的内容推荐给你的朋友、同事们，我们下一讲见。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 7 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 17 | 实战痛点3：Vue 3 中如何集成第三方框架

下一篇 加餐01 | 什么是好的项目？

更多课程推荐

精选留言 (11)

写留言

海阔天空

2021-11-26

性能优化是一个讲不完的话题。以前项目做性能优化就是 按需优化。当然老师讲的，性能优化的一个重要原则，是不要过度优化。非常好，给了一个参考的方向。

3

II

2021-11-26

性能优化这个问题，首先要知道问题是怎么是怎么产生的？而这一切的根源其实包含在“从输入URL到页面生成发生了什么？”这个问题中，感兴趣的可以参考《浏览器工作原理与实践》，导航、渲染、页面等章节。
要想性能优化好，底层原理少不了。

展开

作者回复: cool，浏览器原理的课也贼棒 强推

2

奇奇
2021-11-29

为什么我的是在项目的跟目录多了一个 stats.html ，并不是在 build 后 dist 中生成呢

1

Geek_d47f78
2021-11-28

支持十几个G的文件上传，断点续传，分片也行吗？

展开

共 1 条评论 >

1

Danny
2021-11-27

dns-prefetch 怎么让打包后的代码中添加这个呢

展开

1

南山
2021-11-26

用户体验的优化：比如增加骨架屏
打包优化：按需加载和懒加载

1

润培
2021-11-26

虽说Vue3底层做了很多优化，但很多业务逻辑还是我们自己实现的，例如文中提到的斐波那契数列。
这部分的优化更依赖于我们的算法能力。

展开

1

@
2021-11-26

这时候我们就需要用懒加载和按需加载的方式，去优化项目整体的体积。
具体实操呢，想知道解决办法

展开

1

Geek_24e4e0
2021-12-04

Composition API + <script setup>

```
 const { product } = defineProps<{
    product: Product;
  }>();
```

...

展开

Geek_a2f5d8

2021-12-03

u1s1，配图还是很有品味的

展开

大导演

2021-11-28

router-view v-slot 做转场动画，提示 template 须是一个根元素，用 div 包裹后，动画正常，但是获取不到 body 的高度了 getBoundClientRect 只返回了 width right 的值，其他值都是 0。设置 template 是多节点，就可以获取到 height。一直找不到原因....

展开